REMARKS

Claims 1-35 and 37-44 remain pending with claims 1, 16, 22, 29, and 40 being independent.

The Examiner stated that only claims 1-28 are pending. However, as shown in the attached copy of the preliminary amendment filed with the continuation, claims 29-44 had been previously added.

The remainder of the remarks follow a copy of the Office Action text in small, bolded letters.

2. Claims 1-15, and 22-28 of the instant application are rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over some claims of U.S. Patent No. 6,625,654

Applicants will file a terminal disclaimer before receiving a notice of allowance.

20. Claims 1-12, 16-19, and 22-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over anticipated by Kahle et al (Hereafter, Kahle), U.S. Pat. No. 6,212,542 in view of Change et al (Hereafter, Chang), U.S. Pat. No. 6,338,078.

21. Regarding claims 1, 16, and 22, Kahle teaches a method for packet (eg., data processing (col. 4, lines 26-28) comprising operations on the packets (eg. data) with plurality of program threads to affect processing of the packets (Abstract; Figure 4; col. 10, lines 25-56).

Kahle does not explicitly teach the packet is receiving from the network. However, Kahle does suggest that the Kahle's invention can be implemented as a computer program product for use with a data processing system, such as a computer network (col. 27, lines 8-22).

Chang, in the same field of data processing, discloses a packet ,which is received from the network (Abstract). It would have been obvious to one of ordinary skill in the art at the time of invention to incorporate a Chang's teaching into Kahle's method to process a network packet to improve network throughput and take advantage of Multiple processors scalability.

As amended, claim 1 recites "operating on network packets with a plurality of [the] program threads to affect processing of the packets". The threads are provided by "a processor having multiple engines.. each of the multiple engines having multiple program counters for different program threads provided by the respective engine". For example, assuming 8-engines and 4-threads/program counters per engine, a processor could provide 32 concurrently on-going threads.

Kahle does not describe "multiple engines having multiple program counters for different program threads". Briefly, Kahle describes a multi-scalar processor where a single program is decomposed into multiple tasks that are, in turn, assigned to processing elements (PEs) (col. 2, line 64 - col. 3, line 7) (e.g., PE's 132-138 in FIG. 4 of Kahle). Each PE can service a single assigned sub-portion of the program at a time (see, e.g., col. 17 line 13- col. 18, line 50). Thus, for example, since FIG. 4 has four PEs, the multi-scalar processor shown in FIG. 4 can service up to four portions of the program concurrently, one task by each of the four PEs. Importantly, however, a given PE only processes a single task at a time. Thus, individual processing elements do not include multiple program counters for multiple threads.

The Applicant's also disagree that it would have been obvious to combine Kahle and Chang. Again, Kahle's multi-scalar processor divides a single program into smaller tasks for executing by the processing elements. Chang queues packets belonging to different connections (e.g., TCP/IP connections) to different processors (see FIG. 3). However, queuing packets belonging to different flows to different processors does not make sense in Kahle. That is, in Kahle, the collection of processing elements that make up the multi-scalar processor collectively act as a single processor executing a single program. In other words, there is no reason provided to engineer a modification of Kahle to include multiple processor queues for different packet flows when, effectively, only a single processor will process all of the queued packets.

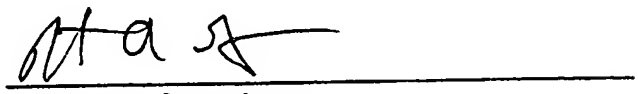
Applicant asks the Examiner to withdraw the rejections of the claims for at least the reasons above.

Docket No.: 10559-139002
Intel Docket No.: P7878D

Applicants have included the fee for a one month extension of time. No other fees are believed due. However, if any other fees are due, please apply such fees to Deposit Account No. 06-1050 referencing attorney docket number: 10559-139002.

Respectfully submitted,

Date: 1/3/04



Robert A. Greenberg
Reg. No. 44,133

ATTORNEYS FOR INTEL
Fish & Richardson P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: Gilbert Wolrich et al.	Art Unit	: 2143
Serial No.	: Unknown	Examiner	: Unknown
Filed	: Herein	Assignee	: Intel Corporation
Title	: THREAD SIGNALING IN MULTI-THREADED NETWORK PROCESSOR		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

PRELIMINARY AMENDMENT

Prior to examination, please amend the application referenced above as follows:

Applicant asks that all claims be allowed in view of the amendment to the claims and remarks contained on the following sheets, a total of 12 pages.

Enclosed is a check for the additional claims. If any other fees are due please apply to Deposit Account No. 06-1050 referencing attorney docket number: 10559-139002.

Respectfully submitted,

Date: 7/8/03

Denis G. Maloney
Reg. No. 29,670

Attorneys for Intel Corporation
Fish & Richardson P.C.
225 Franklin Street
Boston, MA 02110-2804
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 331002251 US

July 8, 2003
Date of Deposit

IN THE SPECIFICATION:

Please amend the specification as follows.

Please insert the following paragraph after the title:

This application is a continuation of, and claims priority to, U.S. Patent Application Serial No. 09/473,799, filed 12/28/1999, and entitled "THREAD SIGNALING IN MULTI-THREADED NETWORK PROCESSOR".

Please replace the paragraph beginning at page 5, line 8 with the following rewritten paragraph:

-- Hardware context swapping enables other contexts with unique program counters to execute in the same microengine. Hardware context swapping also synchronizes completion of tasks. For example, two program threads could request the same shared resource e.g., SRAM. Each one of these separate functional units, e.g., the FBUS interface 28, the SRAM controller 26b ~~26a~~, and the SDRAM controller 26a ~~26b~~, when they complete a requested task from one of the microengine program thread contexts reports back a flag signaling completion of an operation. When the flag is received by the microengine, the microengine can determine which program thread to turn on. --

Please replace the paragraph beginning at page 7, line 9 with the following rewritten paragraph:

Referring to FIG. 2, each of the microengines 22a-22f includes an arbiter that examines flags to determine the available program threads to be operated upon. Any program thread from any of the microengines 22a-22f can access the SDRAM controller 26a, SRAM ~~SDRAM~~ controller 26b or FBUS interface 28. The SDRAM controller 26a and SDRAM controller 26b each include a plurality of queues to store outstanding memory reference requests. The queues either maintain order of memory references or arrange memory references to optimize memory bandwidth.

Please replace the paragraph beginning at page 11, line 5 with the following rewritten paragraph:

Referring to FIG. 3, an exemplary one of the microengines 22a-22f, e.g., microengine 22f is shown. The microengine includes a control store 70 which, in one implementation, includes a RAM of here 1,024 words of 32 bits. The RAM stores a microprogram that is loadable by the core processor 20. The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-72c ~~72a-72d~~. The four micro program counters 72a-72c ~~72a-72d~~ are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ_#_EVENT_RESPONSE; FBI_EVENT_RESPONSE; SRAM _EVENT_RESPONSE; SDRAM _EVENT_RESPONSE; and ASB _EVENT_RESPONSE) from each one of the shared resources, e.g., SRAM 26b ~~26a~~, SDRAM 26a ~~26b~~, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a program thread has completed and signaled completion, the program thread needs to wait for that completion signal, and if the program thread is enabled to operate, then the program thread is placed on an available program thread list (not shown). The microengine 22f can have a maximum of, e.g., 4 program threads available.

IN THE CLAIMS:

Please amend the claims as follows:

1. (Currently Amended) A method for network packet processing comprises:

receiving network packets at a network processor having multiple engines collectively executing multiple program threads; and

operating on the network packets with a plurality of the program threads to ~~affect processing~~ process the packets.

2. (Original) The method of claim 1 wherein operating comprises:

using at least one program thread to inspect a header portion of the packet.

3. (Original) The method of claim 2 wherein operating further comprises:

signaling by the at least one program thread that a packet header has been processed.

4. (Currently Amended) The method of claim 1, wherein the plurality of program threads comprise ~~are~~ a scheduler program ~~threads~~ thread to schedule task orders for processing and a processing program ~~threads~~ thread that ~~process~~ processes packets in accordance with task assignments assigned by the scheduler program thread ~~threads~~.

5. (Currently Amended) The method of claim 1 wherein ~~each~~ the program threads ~~thread~~ write a message to a register that indicates ~~its~~ the program threads' current ~~status~~ statuses.

6. (Currently Amended) The method of claim 5 wherein interpretation of the message is fixed by a software convention determined between a scheduler program thread and a processing program ~~threads~~ thread called by the scheduler program thread.

7. (Original) The method of claim 5 wherein status messages include busy, not busy, not busy but waiting.

8. (Original) The method of claim 5 wherein a status message includes not busy, but waiting and wherein the status of not busy, but waiting signals that the current program thread has completed processing of a portion of a packet and is expected to be assigned to perform a subsequent task on the packet when data is made available to continue processing of the program thread.

9. (Currently Amended) The method of claim 5 wherein the register is a globally accessible register that can be read from or written to by ~~all current~~ different ones of the program threads executed by different ones of the multiple engines.

10. (Original) The method of claim 4 wherein scheduler program threads can schedule any one of a plurality of processing program threads to handle processing of a task.

11. (Original) The method of claim 10 wherein the scheduler program thread writes a register with an address corresponding to a location of data for the plurality of processing program threads.

12. (Original) The method of claim 11 wherein a selected one of the plurality of processing program threads that can handle the task reads the register to obtain the location of the data.

13. (Original) The method of claim 12 wherein the selected one of the plurality of processing program threads reads the register to obtain the location of the data and to assign itself to processing the task requested by the scheduler program thread.

14. (Original) The method of claim 12 wherein the selected one of the plurality of processing tasks reads the register to obtain the location of the data, while the register is cleared by reading the register by the program thread to assign itself to process the task.

15. (Original) The method of claim 13 wherein when another one of the plurality of processing program threads assignable to the task attempts to read the register after it has been cleared, it is provided with a null value that indicates that there is no task currently assignable to the processing program thread.

16. (Currently Amended) A parallel hardware-based multithreaded processor for receiving network packets comprises:

a general purpose processor that coordinates system functions; and

a plurality of microengines that support multiple program threads, and operate on network packets with a plurality of program threads to ~~affect processing~~ process the packets.

17. (Currently Amended)) The processor of claim 16 wherein one of the plurality of microengines executes scheduler program threads and ~~remaining~~ other ones of the microengines execute processing program threads.

18. (Original) The processor of claim 16 further comprising a global thread status register wherein each program thread writes a message to the global status register that indicates its current status.

19. (Original) The processor of claim 18 wherein interpretation of the message is fixed by a software convention determined between a scheduler program thread and processing program threads called by the scheduler program thread.

20. (Original) The processor of claim 16 further comprising:

a read once register, wherein the scheduler program thread writes the read once register with an address corresponding to a location of data for the plurality of processing program threads and when a selected one of the plurality of processing program threads reads the register to obtain the location of the data, assigns itself to processing the task requested by the scheduler program thread, while the register is cleared by reading the register by the program thread.

21. (Original) The processor of claim 20 wherein when another one of the plurality of processing program threads assignable to the task attempts to read the read once register after it has been cleared, it is provided with a null value that indicates that there is no task currently assignable to the processing program thread.

22. (Currently Amended) An apparatus comprising a machine-readable storage medium having executable instructions for network processing, the instructions enabling the apparatus to:

receive network packets; and

operate on the network packets with a plurality of program threads collectively provided by multiple engines of a network processor to affect processing of the packets.

23. (Original) The apparatus of claim 22 wherein instructions to operate further comprise instructions to:

use at least one program thread to inspect a header portion of the packet.

24. (Original) The apparatus of claim 22 further comprising instructions to provide scheduler program threads to schedule task orders for processing and processing program threads to process packets in accordance with task assignments assigned by the scheduler program threads.

25. (Original) The apparatus of claim 22 wherein each program thread writes a message to a register that indicates its current status.

26. (Original) The apparatus of claim 25 wherein the register is a globally accessible register that can be read from or written to by all current program threads.

27. (Original) The apparatus of claim 22 wherein the scheduler program thread writes a register with an address corresponding to a location of data for the plurality of processing program threads and a selected one of the plurality of processing program threads that can handle the task reads the register to obtain the location of the data, and clears the register after reading by the program thread.

28. (Original) The apparatus of claim 27 wherein when another one of the plurality of processing program threads assignable to the task attempts to read the register after it has been cleared, it is provided with a null value that indicates that there is no task currently assignable to the processing program thread.

29. (New). A system, comprising:

multiple engines of a network processor to execute instructions of multiple threads to process packets received via at least one network interface; and

at least one storage element shared by different ones of the multiple threads to provide inter-thread communication between the threads.

30 (New). The system of claim 29, wherein the at least one storage element comprises a storage element having associated logic to set the register to a reset value after the register is read.

31 (New). The system of claim 30, further comprising:

instructions of a first of the multiple threads, disposed on a computer readable medium, to write a value to the storage element identifying availability of a task; and

instructions of a second of the multiple threads, disposed on a computer readable medium, to read the storage element and perform the task if the retrieved value of the storage element does not equal the reset value of the storage element.

32 (New). The system of claim 29, wherein the at least one storage element comprises a register having different portions allocated to different threads of the different engines.

33 (New). The system of claim 32, further comprising

instructions of a first of the multiple threads, disposed on a computer readable medium, to write a task completion status to the portion allocated to the first of the multiple threads; and

instructions of a second of the multiple threads, disposed on a computer readable medium, to read the task completion status of a portion allocated to the first of the multiple threads.

34 (New). The system of claim 29, further comprising:

instructions of a first of the multiple threads, disposed on a computer readable medium, to set a bit in an array to identify a packet added to a queue.

35 (New). The system of claim 34, further comprising:

instructions of a second of the multiple threads, disposed on a computer readable medium, to determine a queue to service based on the array.

36 (New). The system of claim 29, wherein at least one of the multiple engines comprises an engine having hardware to store execution contexts of multiple ones of the multiple threads.

37 (New). The system of claim 29, further comprising:

instructions of a scheduler thread, disposed on a computer readable medium, to assign a different thread to process a received packet.

38 (New). The system of claim 29, further comprising:

instructions of a receive thread, disposed on a computer readable medium, to reassemble a packet assigned to the thread.

39 (New). The system of claim 29, wherein the storage element comprises a memory internal to the network processor shared by the different engines.

40 (New). A system, comprising:

at least one Ethernet media access controller; and

a network processor communicatively coupled to the at least one Ethernet media access controller to process packets received via the at least one Ethernet media access controller, the processor comprising:

multiple engines to execute instructions of multiple threads, multiple ones of the multiple engines having hardware to store the execution contexts of multiple ones of the multiple threads;

at least one storage element to provide inter-thread communication between the threads; and

thread instructions, disposed on a computer readable medium, to cause at least one of the multiple engines to:

identify a thread to process a received packet;

set a bit within an array to identify a queue for the received packet; and

access the array to identify a queue to service.

41. (New) The system of claim 40, wherein the storage element comprises a memory internal to the network processor, the memory being shared by multiple ones of the multiple engines.

42. (New) The system of claim 40, wherein the storage element comprises at least one of the following: a register having portions assigned to the different threads and a register that resets its value in response to a read access.

43. (New) The method of claim 1, wherein at least one of the engines comprises an engine to execute multiple ones of the multiple threads.

44. (New) The apparatus of claim 22, wherein at least one of the engines comprises an engine to execute multiple ones of the multiple threads.

REMARKS

Applicants present claims 1-44 for consideration with claims 1, 16, 22, 29, and 40 being independent.

Applicant's undersigned attorney can be reached by telephone at the number shown above.